

FEATURES

14 A Digital Filtering Primer

20 Spectral Analysis

40 Introduction to
Doremi-DSP

50 Fast-scaling Routine for
Floating-point RISC and
DSP Processors

A Digital Filtering Primer

You'll often find articles written about digital filters, how they work, and how to design them. But, how often do they approach the subject from a practical standpoint? Tom shares some design tips from the trenches.

FEATURE ARTICLE

Tom Ulrich

a common maxim for designing embedded controllers is that a controller is no better than its feedback sensor. For example, if you are trying to control the position of something, a controller can do no better than its sensor's ability to measure a position. You can have the hottest microprocessor or DSP in the world, but if you can't accurately sense what you are trying to control, you will get poor results.

But, what if you are stuck using a sensor that is noisy or a few bits short of resolution? Is there anything you can do?

"Yes!"

The key is to use the processor to enhance the data before using it to control the data. And, the best part is there are simple techniques that enable you to do this even with a low-performance processor.

If you need this kind of information, I invite you to join me on a journey into the world of digital filtering. We'll take a look at how digital filters work, important details to remember when using digital filters, and implementation tips including sample code from real engineering projects.

THE BASICS

The most common digital filtering technique is to simply take a running average of several samples of data. The idea is that rather than just reading the transducer each time you close your control loop, you read it every time you take a piece of data and average it into the previous value using a weighting factor. In the process, the

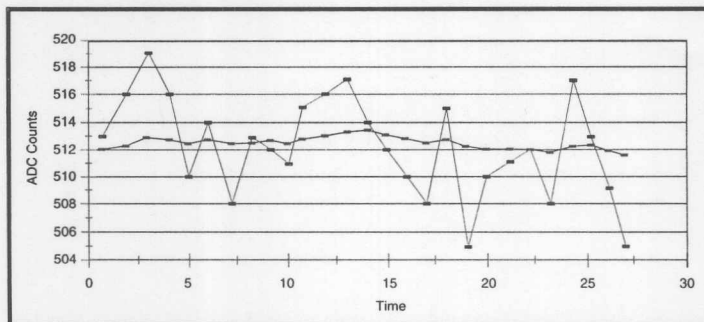


Figure 1—A simple digital filter produces the same result as a basic analog RC filter, eliminating high-frequency random noise.

data becomes less noisy since random errors tend to cancel. Mathematically, this is expressed as:

$$X_{\text{filtered_new}} = K \times X_{\text{filtered_old}} + (1 - K) \times X_{\text{raw}}$$

where $X_{\text{filtered_new}}$ is the latest filtered value, $X_{\text{filtered_old}}$, the previous filtered value, X_{raw} , the value just read from the sensor, and K , the filter constant (this always has a value between 0 and 1 in which 0 represents no filtering and 1 involves total filtering). To minimize the number of multiplication operations, this equation is usually implemented as:

$$X_{\text{filtered_new}} = X_{\text{raw}} + K \times (X_{\text{filtered_old}} - X_{\text{raw}})$$

This technique gives a filter with much the same characteristics of a simple RC filter. Figure 1, which shows some raw "noisy" data read in from a sensor and the filtered result, illustrates the effect of this equation.

In looking at Figure 1, you may notice another interesting thing about digital filtering. The filtered signals are fractional values of the analog-to-digital converter's (ADC) codes, which means they are at a higher resolution than the nonfiltered signal. In fact, using digital filtering often gives you the equivalent of one or two additional bits on your ADC! This phenomenon occurs because the filter averages out the white noise on your system.

For example, suppose you have a voltage of 5.05 V on an ADC which was scaled from 0 to 10 V. If the signal was perfect, the ADC would always return a value of 129. However, if

there was one bit (about 0.04 V) of white noise on the signal, it usually returns 129 with occasional values of 130 and 128. If the noise is truly white (a fairly good assumption), we would find that the occurrences of the other values would alter the filtered value to be 129.25, a resolution you normally need a 10-bit ADC to obtain.

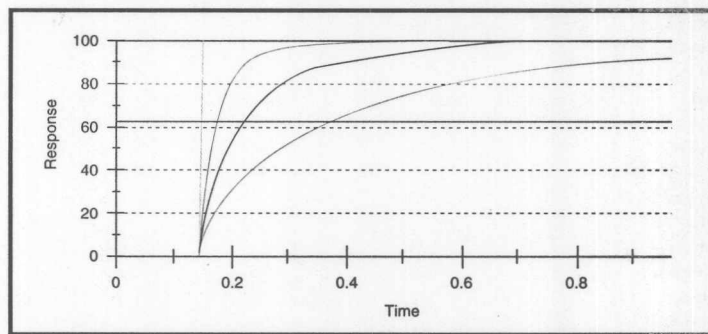


Figure 2—Digitally filtering a step function with three different filter constants produces slightly different responses.

To further illustrate this technique, Figure 2 shows the same filtering scheme with three different filter constants on a simple step function. Notice that on this graph, I have drawn a line showing the one-time-constant response ($1 - e^{-1} = 0.63$). Using a spreadsheet to model the filter with a step function is an easy way to determine the time constant. Table 1 shows the time constants corresponding to the three filter constants used in Figure 2.

Figure 3 shows the same filter constants applied to a simple sine wave. This graph clearly shows that a phase lag along with significant

attenuation are introduced by digital filtering, as with any type of filtering. Table 2 shows the actual phase lags as determined again from a simple spreadsheet model of the filter and response.

IMPORTANT DETAILS TO REMEMBER

Now that we have looked at how a digital filter works, we need to look at some details that are important to know, but that textbooks usually forget to mention.

- The time constant needs tuning.

When you use a digital filter, the time constant becomes an additional item to tune. For example, if you use digital filtering to clean up data used in a PID servo loop, you will need to tune the filter constant as well as the P, I, and D gains. Furthermore, since the actual time constant of the filter is a function of both the filter constant K

and the sample rate time, you need to consider filtering requirements as well as PID requirements.

Although it may appear from first impressions that digital filtering can be more trouble than it is worth, the bottom line is that sometimes you can't get adequate stability without it.

K	Sample Time	Time Constant
10	0.01	0.40 s
25	0.01	0.25 s
45	0.01	0.20 s

Table 1—The time constant of the filter whose response is shown in Figure 2 decreases as the filter constant increases.

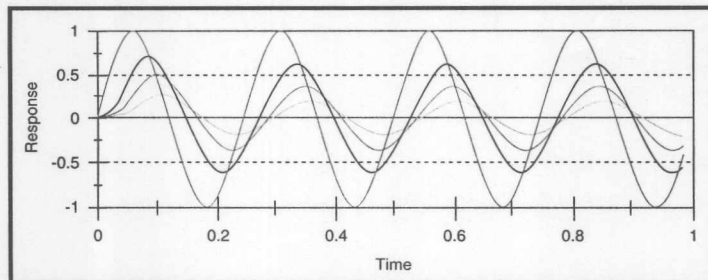


Figure 3—Digitally filtering a pure sine wave not only attenuates the signal amplitude, but also shifts its phase (like any analog filter would).

With it, you have to work hard to tune the system, but an acceptable solution is possible.

- Filtering introduces a lag into your control system.

Remembering the lag is especially important if the system dynamics require the use of lead terms such as derivative (or rate) gain. You must be careful not to nullify the advantage of lead terms by using too much filtering. There is a delicate balance that even the most sophisticated control engineers struggle with, but a balance between the two extremes does exist.

When writing the software for Parker's original electrohydraulic actuator (EHA), I was able to filter both the position and velocity terms without killing the effect of the acceleration gain. In that case, the acceleration term was doubly filtered, but still able to contribute a significant leading effect. It was a difficult tuning task (and I had help from a controls guy), but without it we could not get adequate response from our controller.

- You still need an analog filter.

If you have a signal with noise at a frequency higher than the frequency at which you are sampling the data, you can get a phenomenon called *aliasing*. With aliasing, as you sample the higher-frequency data, you can end up reading "beat frequencies," which appear as lower-frequency signals.

For example, suppose we have an unshielded pressure sensor line that is picking up noise from fluorescent lights driven off a 60-Hz AC line. Let's further suppose that we are sampling

data at 25 Hz. Here the problem stems from the fact that, at 25 Hz, we are not sampling the whole wave. The filtering is smoothing misrepresentative data points into a fictitious waveform.

The bottom line: anytime you use digital filtering, you must have an analog filter on your signal inputs to

K	Sample Time	Phase Lag
10	0.01	72.0°
25	0.01	57.6°
45	0.01	43.2°

Table 2—Increasing the signal attenuation by decreasing the filter constant also increases the phase lag (as shown in Figure 3).

filter out higher frequency noise. So, for instance, on the Parker EMC100 digital-programmable motion controller, I used an analog RC antialiasing filter at 600 Hz for a signal that I sampled at 1000 Hz.

A question sometimes raised at this point is, "If you always need a

hardware filter when using a software filter, why not just forget the software filter and do it all in hardware?"

There are two reasons to not rely solely on hardware. First, implementing a high-frequency antialiasing filter requires only a small (and inexpensive) capacitor and resistor. But, to implement lower-frequency filters, you need much larger (and more expensive) capacitors. Hence, it is usually more cost effective to implement lower-frequency filters in software.

Second, frequently the selection of proper time constants for these filters is a matter of tuning. For different installations, you might want different time constants. With a software filter, adjusting a time constant is no more painful than adjusting a gain. But with a hardware filter, you've got to get out the soldering iron and change capacitors or resistors to make a time-constant change.

- Remember to initialize the filter.

A common mistake in implementing a digital filter is failing to properly initialize the running average. Sometimes this mistake arises in the form of simply forgetting to initialize the average at all. Other times, it takes the form of initializing to zero.

The proper approach is to initialize the average to a value near the true value so the filter doesn't have to deal with what is, in effect, a big step function at powerup. A common way to initialize the average is to read the sensor one time at powerup. The

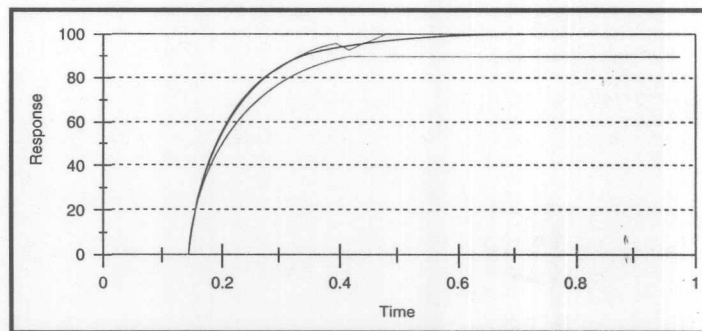


Figure 4—Filtering a step function using floating-point math in the filter routine produces a nice, smooth output. Using integer math and keeping track of remainders results in some bumps, but no DC offset. Using integer math but dropping the remainders produces a DC offset in the output.

Listing 1—Implementing a digital filter in C requires little code.

```
void filter(int *filtered, int raw, unsigned int filtConstant,
           unsigned int *low)
{
    long along;

    /* convert to long to avoid overflow on multiply */
    along = (*filtered - raw);
    along = (along * filtConstant);

    /* add remainder from last time */
    along += *low;

    /* store remainder for next time through */
    *low = 0x0000ffff & along;

    /* shift right for fractional filter constant */
    along = along >> 16;
    *filtered = raw + along;
}
```

reading is then used as the value which initializes the average.

A purist may want to initialize the sum to the average of two or three readings, but that is usually not necessary unless your system is extremely noisy. The goal is to get the value nearly right to avoid an extreme response to a step function; the initial value doesn't have to be perfect, just close.

IMPLEMENTATION TIPS

Now that we have looked at how a digital filter works and some important details to remember, we need to look at some implementation tips.

- Don't use floating-point math.

Unless you have the very unusual situation of having an embedded controller with ample horsepower and resources, the last thing you want to do is use floating-point math with this equation. Instead, use fractional-integer math.

You want to represent a noninteger number as some fractional value of either 256 or 65,536. For instance, if you have a 16-bit controller, the natural way to represent the fraction 1/2 is with the number 32,768. To multiply, you multiply the number by the constant and shift it by 16 when you are all done.

For example, suppose the filter constant is 0.25, our running average is

2000, and the new value is 1900. K would equal 65,536 divided by 4 or 16,384. Hence, the equation is:

$$X_{\text{filtered}} = ((16384 \times (2000 - 1900)) \gg 16) + 1900 = 1925$$

Using fractional-integer math rather than floating-point math can easily reduce computation time by an order of magnitude.

- Use an integer and a remainder, rather than a long integer number.

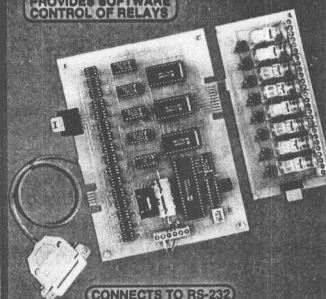
For a 12-bit ADC, you will probably find that using only 16 bits of filtered data will not give you enough resolution and will actually introduce truncation errors in your filtered value. But, if you opt for using a 32-bit word for your filtered data when running on a 16- or 8-bit microprocessor, you greatly increase the processing time needed to do the multiply.

The trick is to hold on to the remainder from the previous pass with the filter. (With a shift operation, the remainder is the part that gets shifted away when you divide by 256 or 65,536 as described above.) Each time you do the multiply, add the remainder from the previous pass and then store the new remainder.

Using a remainder, rather than a longer word length, also offers the advantage of using 16, not 32, bits for subsequent calculations when you use the filtered data in something like a

RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

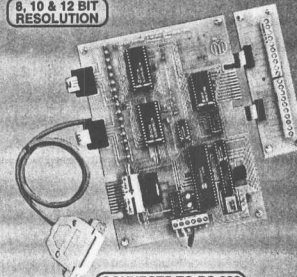


CONNECTS TO RS-232

AR-16 RELAY INTERFACE (16 channel).....\$ 99.95
Two 8 channel (TTL level) outputs are provided for connection to relay cards or other devices (expandable to 128 relays using EX-16 expansion cards). A variety of relays cards and relays are stocked. Call for more info.
AR-2 RELAY INTERFACE (2 relays, 10 amp).....\$ 44.95
RD-8 REED RELAY CARD (8 relays, 10 VA).....\$ 49.95
RH-8 RELAY CARD (10 amp SPDT, 277 VAC).....\$ 69.95

ANALOG TO DIGITAL

8, 10 & 12 BIT RESOLUTION



CONNECTS TO RS-232

ADC-16 A/D CONVERTER* (16 channel/8 bit).....\$ 99.95
ADC-8G A/D CONVERTER* (8 channel/10 bit).....\$124.90
Input voltage, ampereage, pressure, energy usage, joysticks and a wide variety of other types of analog signals. RS-422/RS-485 available (lengths to 4,000'). Call for info on other A/D configurations and 12 bit converters (terminal block and cable sold separately).
ADC-8E TEMPERATURE INTERFACE* (8 ch.).....\$ 139.95
Includes term. block & 8 temp. sensors (-40° to 145° F).
STA-8 DIGITAL INTERFACE* (8 channel).....\$ 99.95
Input on/off status of relays, switches, HVAC equipment, security devices, smoke detectors, and other devices.
STA-8D TOUCH TONE INTERFACE*.....\$ 134.90
Allows callers to select control functions from any phone.
PS-4 PORT SELECTOR (4 channels RS-422).....\$ 79.95
Converts an RS-232 port into 4 selectable RS-422 ports.
CO-485 (RS-232 to RS-422/RS-485 converter).....\$ 44.95

*EXPANDABLE...expand your interface to control and monitor up to 512 relays, up to 576 digital inputs, up to 128 analog inputs or up to 128 temperature inputs using the PS-4, EX-16, ST-32 & AD-16 expansion cards.

•FULL TECHNICAL SUPPORT...provided over the telephone by our staff. Technical reference & disk including test software & programming examples in Basic, C and assembly are provided with each order.

•HIGH RELIABILITY...engineered for continuous 24 hour industrial applications with 10 years of proven performance in the energy management field.

•CONNECTS TO RS-232, RS-422 or RS-485...use with IBM and compatibles, Mac and most computers. All standard baud rates and protocols (50 to 19,200 baud). Use our 800 number to order FREE INFORMATION PACKET. Technical information (614) 464-4470.

24 HOUR ORDER LINE (800) 842-7714
Visa-Mastercard-American Express-COD

International & Domestic FAX (614) 464-9656
Use for information, technical support & orders.

ELECTRONIC ENERGY CONTROL, INC.
380 South Fifth Street, Suite 604
Columbus, Ohio 43215-5438

Listing 2—By using a bit of inline assembler (in this case for an 80C196 microcontroller) to replace some inefficient code generated by the compiler, the filter program runs 70% faster.

```

register int diff;
register long prod;

void filter(int *filtered, int raw, unsigned int filtConstant,
            unsigned int *low)
{
    diff = (*filtered - raw); /* filter temperature */
    asm MUL prod, diff, filtConstant; /* prod=diff * filtConstant */
    prod += *low; /* add in low word left from last time */
    *low = prod; /* store low word for next time */
    asm SHRAL prod, #15;
    asm ADD prod, raw;
    *filtered = prod;
}

```

PID algorithm. This technique can easily reduce computation time by a factor of 2–3 times.

Figure 4 shows sample results of this technique. In this figure, you see three filtered results: a result using a floating-point (that's the perfect looking one), a result using a remainder technique (the one with a few bumps, but no DC offset), and a result

using integer math without remainders (that's the one with the DC offset).

Listing 1 offers an example of a real implementation of such a filter. The program includes the original C code used to implement a digital filter on the Parker-Hannifin Vapor-Cycle-System Digital Controller for the AH64D Apache Longbow Helicopter.

Note how I take the difference between the filtered and new values, then place the result in a long real number called a long. This is important because otherwise the C compiler assumes I want an integer result for the subsequent multiply and chops off the high word.

• Use inline assembler when time is tight.

Listing 2 contains the code of Listing 1, except that it is rewritten for increased performance. I used some inline assembler to make the code smarter than that generated by the compiler.

The compiler implemented the multiplication by multiplying a long by a long, which means it did four 16-bit multiplication operations ($MSW1 \times MSW2$, $MSW1 \times LSW2$, $LSW1 \times MSW2$, and $LSW1 \times LSW2$) and then added the products together. In fact, all it needed to do was simply multiply $LSW1 \times LSW2$ with no addition afterwards. By explicitly doing the multiply in assembler, I reduced the execution time of this module by 70%. Further time was saved using registers for some of the intermediate results and by using assembler again to do the shift and final assigns.

In summary, I have tried to present the basics of digital filtering along with some important implementation tricks. With these tools, you have all you need to solve your next noisy sensor problems. 

Tom Ulrich received B.S. and M.S. degrees in engineering from the University of California at Irvine and is principal engineer in the Gull electronics system division of Parker Hannifin. He has written embedded software for numerous new products for both industrial and aerospace divisions of Parker. He may be reached at Parker Hannifin, 14300 Alton Parkway, Irvine, CA 92718.

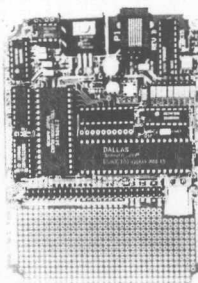
TURBO-128

THE NEXT GENERATION EMBEDDED CONTROLLER

**** NO DEVELOPMENT TOOLS REQUIRED ****

Ready to Program in BASIC or ASSEMBLY

Photronics Research introduces the T-128: A True Single Board BASIC Development System. The T-128 is based on Dallas Semiconductor's new 8051-compatible DS80C320. With its 2X clock speed (25MHz) and 3X cycle efficiency, an instruction can execute in 160ns: an 8051 equivalent speed of 62.5MHz!!! Equally impressive is the T-128's high-speed NVRAM interface. Any of the 128K RAM may be Program Development has never been faster or more convenient, even with the finest EPROM emulator. The T-128 features PORT O bias and EA-select for DS87C520 upgrade.



BASIC-520 ACCELERATED 800%-1000%

PROCESSOR

- Dallas Semiconductor's DS80C320
- 300% more efficient than the 8051
- Three 16-bit Timers/Counters
- 13 Interrupts (6 Ext, 7 Int)
- A second 16-bit Data Pointer
- 384 Bytes of Internal RAM
- Programmable Watchdog
- Brownout Protection
- Power-Fail Reset/Interrupt
- Power-On Reset
- Fully supported by Franklin C51

MEMORY

- Entire 128K Memory Map populated with fast NVRAM (64K DATA • 64K CODE)
- All memory programmed on-board
- Partitionable as CODE/DATA/OVERLAP
- Code Space is Write-Protectable
- State-of-the-Art Data Protection

BASIC-520

- Modified BASIC-52 Interpreter (BASIC-520)
- Now Fast Enough for New Applications
- Stack BASIC Programs and Autotest
- CALL ASM Routines for Maximum Speed

I/O

- Three 8-bit Parallel Ports
- Two Full-Duplex RS232 Serial Ports
- Decoded Device I/O Strobes
- 50-Pin Bus Connector

UPGRADE

- DS87C520 processor (33MHz)
- Instruction cycle: 121ns
- 6.25 MIPS
- 8051 equivalent: 82.5 MHz
- Internal 16K ROM/1K SRAM

Only 5" x 3.3/4" • 500-hole Proto Area • Console/Power connected by a single 4-conductor telephone wire (very convenient)

Photronics Research, Inc.

109 Camille St. • Amite, LA 70422 • (504) 748-9911 • Tech Support (504) 748-7090 • FAX (504) 748-4242

Comes Ready to Run with power adapter/cable assembly. Includes utility diskette with DETAILED TECHNICAL MANUAL. \$199 in qty.

I R S

- 401 Very Useful
- 402 Moderately Useful
- 403 Not Useful